Mixed Criticality Systems, are we ready for tomorrow's platforms?

J. Lopez Trescastro, M. Verhoef

19/10/2016

Mixed Criticality Systems

The complexity of embedded systems is increasing continuously…

- The increment of processor power opens the possibility to integrate a large number of functions in the same execution platform
- Software components with different criticality levels must coexist in a common execution platform

➢ **We need to manage system complexity and reduce development cost**

- We need to protect the critical work when faults occur
- We need to provide efficient resource usage
- We need to provide evidence that the system will behave as expected

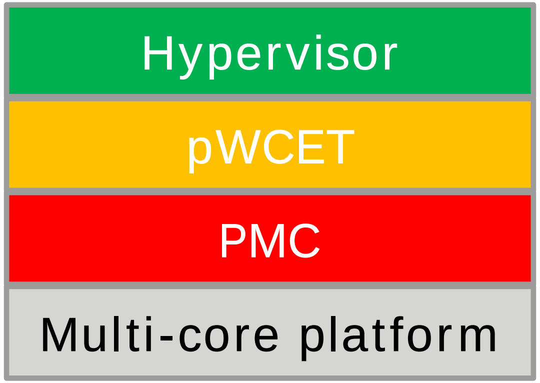
MCS: Multi-Core Platforms

- Tasks running in a multi-core platform can suffer interference **delays from every contending task when accessing a shared resource**
- No WCET technique provides full confidence on execution time bounds for multi-core platforms → Safe techniques are just too pessimistic

- **Probabilistic (pWCET):** can be used to determine the budget required such that the probability of overrunning is below a specified probability.
- **Performance monitoring counters:** can be used to track interference effects arising from contention delays...
    - We can estimate the interference delay that a task can suffer
    - We can detect when a bound is exceeded and **do something about it...**

pWCET + PMC + Hypervisor?

- Good combination of safety and efficient resource usage:
  - The probabilistic approach provides an efficient resource usage
  - The Performance Monitoring Counters provides the "safety net“



- **Can we have a predictable system without (fully) predictable components?**
- **How to deal with COTS components while keeping certification / qualification costs low?**

However…

Low-criticality does **not** mean:

- × No-criticality
- × Soft deadline

→

How to separate higher criticality tasks from the behavior of lower criticality tasks without seriously compromising the RT performance of the latter?

- **Faults are not independent…** -> How can we deal with failure management?
- **Task dependencies…** –> Tasks can not be dropped in an unpredictable way…
  - Do we need fault-trees in our system architecture?
  - How to return to full functionality after a fault?
- How many criticality levels do we need?

Schedulability…

- **Compositional scheduling:** clean isolation of scheduling concerns between partition developers and system integrator -> Inter-partition resource sharing may be difficult to implement…
- **Flat scheduling:** All tasks are considered, independently of the partition where they belong, as a global system -> Very efficient but a change of a task may require the whole system to be reworked…

- How can we achieve the static verification of a system based on all this theory?
  - Is not as simple as dropping low-criticality tasks to make the system schedulable…
  - Which assumptions can we make?
  - To separate or integrate?

Methodology and development tools

- What are still missing?

  Multi-core support / WCET / Hypervisors / Simulators / Compilers / RTOS?

- Incremental certification is a goal to achieve independent certification...
  - How can we achieve incremental certification?
  - Techniques that can guarantee the incremental scheduling of partitions (and efficient resource usage) are still needed

- System modeling: we need to find a way to describe the information relevant for system partitioning and deployment, including Fault Management & Recovery...

- Impact on standards (e.g. ECSS)?

References

- **Mixed Criticality Systems - A Review.** July 2016, Alan Burns and Robert I. Davis

- **Contention-Aware Performance Monitoring Counter Support for Real-Time MPSoCs**. Javier Jalle, Mikel Fernandez, Jaume Abella, Jan Andersson, Mathieu Patte, Luca Fossati, Marco Zulianello, Francisco J. Cazorla

- **Mixed Criticality in Control Systems**. Alfons Crespo, Alejandro Alonso, Marga Marcos, Juan A. de la Puente, Patricia Balbastre